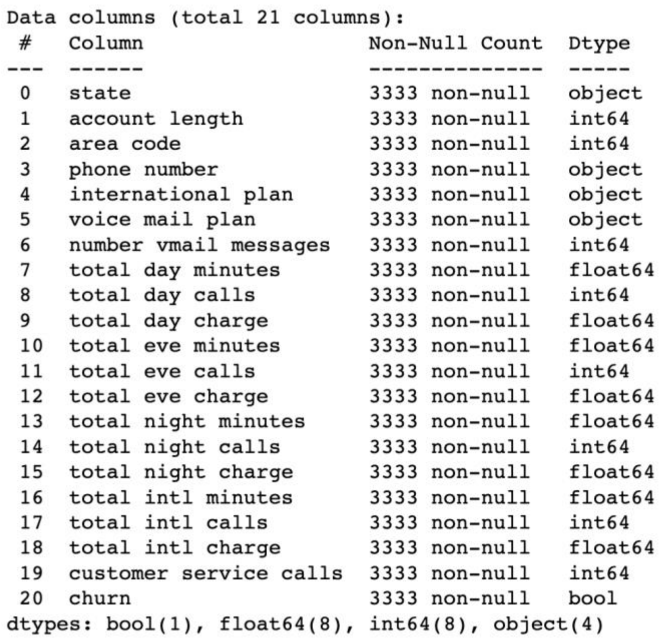
Experiment1:

1. There are 3333 records in the dataset.

2. There are 21 features in dataset.



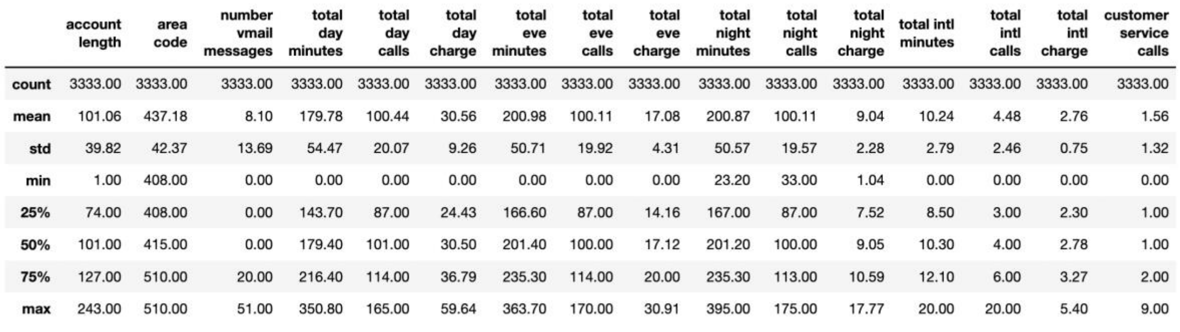
```
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   state                   3333 non-null   object
 1   account length          3333 non-null   int64
 2   area code               3333 non-null   int64
 3   phone number            3333 non-null   object
 4   international plan      3333 non-null   object
 5   voice mail plan         3333 non-null   object
 6   number vmail messages   3333 non-null   int64
 7   total day minutes       3333 non-null   float64
 8   total day calls         3333 non-null   int64
 9   total day charge        3333 non-null   float64
 10  total eve minutes       3333 non-null   float64
 11  total eve calls         3333 non-null   int64
 12  total eve charge        3333 non-null   float64
 13  total night minutes     3333 non-null   float64
 14  total night calls       3333 non-null   int64
 15  total night charge      3333 non-null   float64
 16  total intl minutes      3333 non-null   float64
 17  total intl calls        3333 non-null   int64
 18  total intl charge       3333 non-null   float64
 19  customer service calls  3333 non-null   int64
 20  churn                   3333 non-null   bool
dtypes: bool(1), float64(8), int64(8), object(4)
```

The datatype object and int are discrete, bool is binary and float is continuous.

3. Features like the state, the length of the account, the area code, and the phon e number are not important because there is no correlation between the churn status and these features.

4. No missing values in the dataset because each column contains 3333 observa tions, the same number of rows we observe in the shape.

5. Total day minutes, total day charge, total eve minutes, total eve charge, total n ight minutes, total night charge, total int minutes, and total int charge are some of the continuous numeric features.

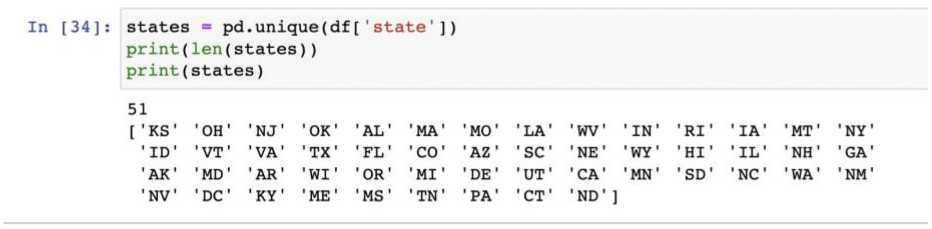


	account length	area code	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge	customer service calls
count	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00
mean	101.06	437.18	8.10	179.78	100.44	30.56	200.98	100.11	17.08	200.87	100.11	9.04	10.24	4.48	2.76	1.56
std	39.82	42.37	13.69	54.47	20.07	9.26	50.71	19.92	4.31	50.57	19.57	2.28	2.79	2.46	0.75	1.32
min	1.00	408.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	23.20	33.00	1.04	0.00	0.00	0.00	0.00
25%	74.00	408.00	0.00	143.70	87.00	24.43	166.60	87.00	14.16	167.00	87.00	7.52	8.50	3.00	2.30	1.00
50%	101.00	415.00	0.00	179.40	101.00	30.50	201.40	100.00	17.12	201.20	100.00	9.05	10.30	4.00	2.78	1.00
75%	127.00	510.00	20.00	216.40	114.00	36.79	235.30	114.00	20.00	235.30	113.00	10.59	12.10	6.00	3.27	2.00
max	243.00	510.00	51.00	350.80	165.00	59.64	363.70	170.00	30.91	395.00	175.00	17.77	20.00	20.00	5.40	9.00

The average, median, maximum, minimum, and standard deviation values can be seen in the above table for the continuous features.

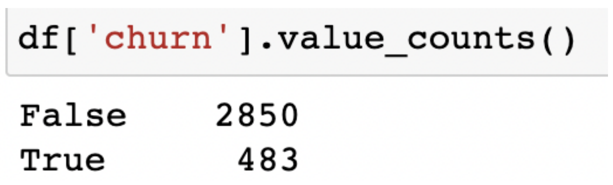
6. The average number of customer service calls a customer makes to the company is 1.56 calls.

7. There are 51 states as described in the below fig.

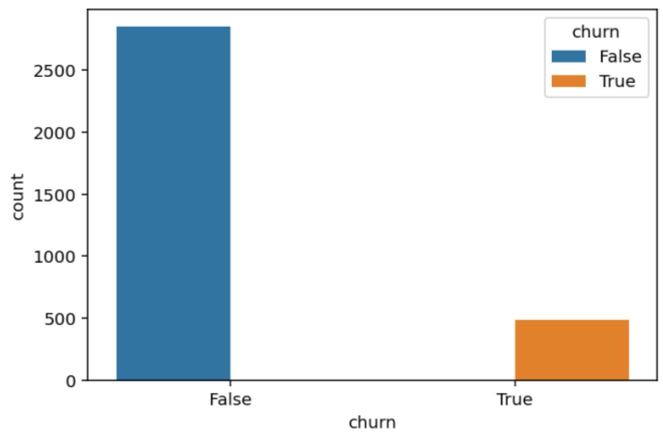


```
In [34]: states = pd.unique(df['state'])
         print(len(states))
         print(states)

         51
         ['KS' 'OH' 'NJ' 'OK' 'AL' 'MA' 'MO' 'LA' 'WV' 'IN' 'RI' 'IA' 'MT' 'NY'
          'ID' 'VT' 'VA' 'TX' 'FL' 'CO' 'AZ' 'SC' 'NE' 'WY' 'HI' 'IL' 'NH' 'GA'
          'AK' 'MD' 'AR' 'WI' 'OR' 'MI' 'DE' 'UT' 'CA' 'MN' 'SD' 'NC' 'WA' 'NM'
          'NV' 'DC' 'KY' 'ME' 'MS' 'TN' 'PA' 'CT' 'ND']
```

8. The following is the distribution of the churn in the dataset.

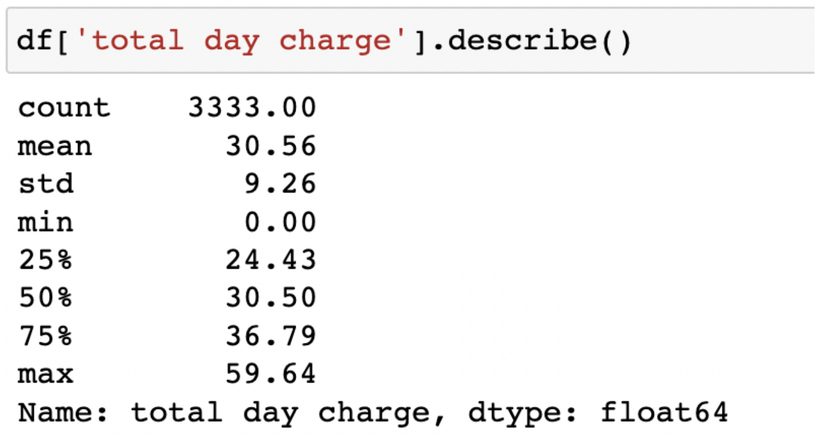


```
df['churn'].value_counts()

False    2850
True      483
```

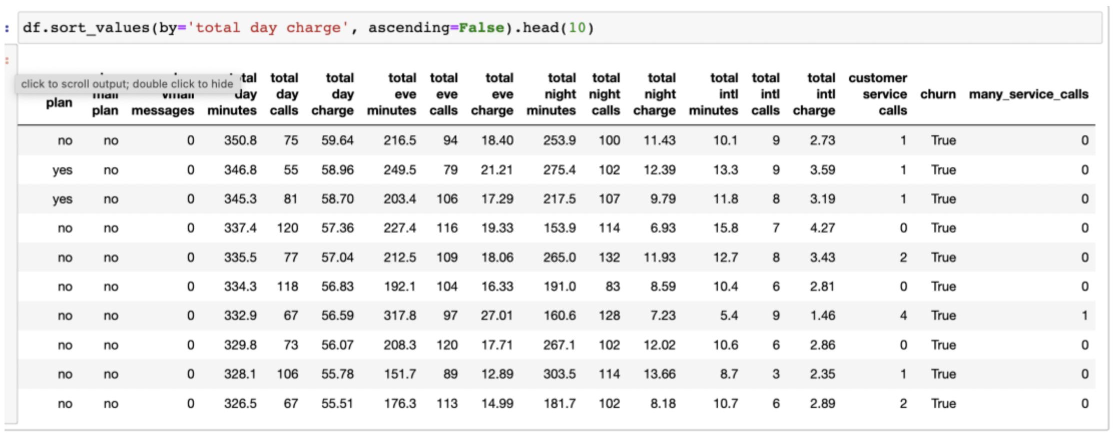


As we can see in the above plot, it is skewed since the true count is very less compared to the false count.

9. The customers' highest and lowest "total day charge" is 59.64 and 0 respectively.



```
df['total day charge'].describe()

count    3333.00
mean       30.56
std         9.26
min         0.00
25%        24.43
50%        30.50
75%        36.79
max        59.64
Name: total day charge, dtype: float64
```

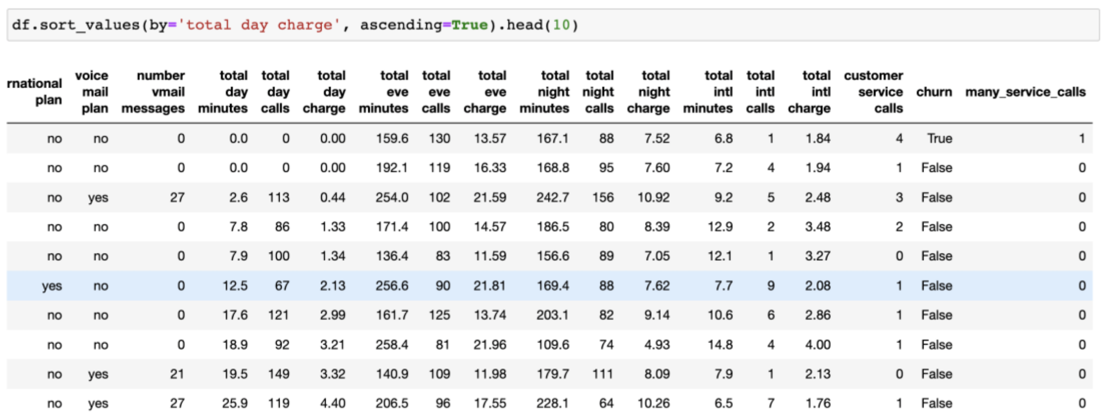


```
df.sort_values(by='total day charge', ascending=False).head(10)
```

click to scroll output; double click to hide

plan	mail plan	vmail messages	tal day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge	customer service calls	churn	many_service_calls
no	no	0	350.8	75	59.64	216.5	94	18.40	253.9	100	11.43	10.1	9	2.73	1	True	0
yes	no	0	346.8	55	58.96	249.5	79	21.21	275.4	102	12.39	13.3	9	3.59	1	True	0
yes	no	0	345.3	81	58.70	203.4	106	17.29	217.5	107	9.79	11.8	8	3.19	1	True	0
no	no	0	337.4	120	57.36	227.4	116	19.33	153.9	114	6.93	15.8	7	4.27	0	True	0
no	no	0	335.5	77	57.04	212.5	109	18.06	265.0	132	11.93	12.7	8	3.43	2	True	0
no	no	0	334.3	118	56.83	192.1	104	16.33	191.0	83	8.59	10.4	6	2.81	0	True	0
no	no	0	332.9	67	56.59	317.8	97	27.01	160.6	128	7.23	5.4	9	1.46	4	True	1
no	no	0	329.8	73	56.07	208.3	120	17.71	267.1	102	12.02	10.6	6	2.86	0	True	0
no	no	0	328.1	106	55.78	151.7	89	12.89	303.5	114	13.66	8.7	3	2.35	1	True	0
no	no	0	326.5	67	55.51	176.3	113	14.99	181.7	102	8.18	10.7	6	2.89	2	True	0

As we can see above, upon sorting in the descending order by "total day charge", we can see that the churn is true is most of the cases.

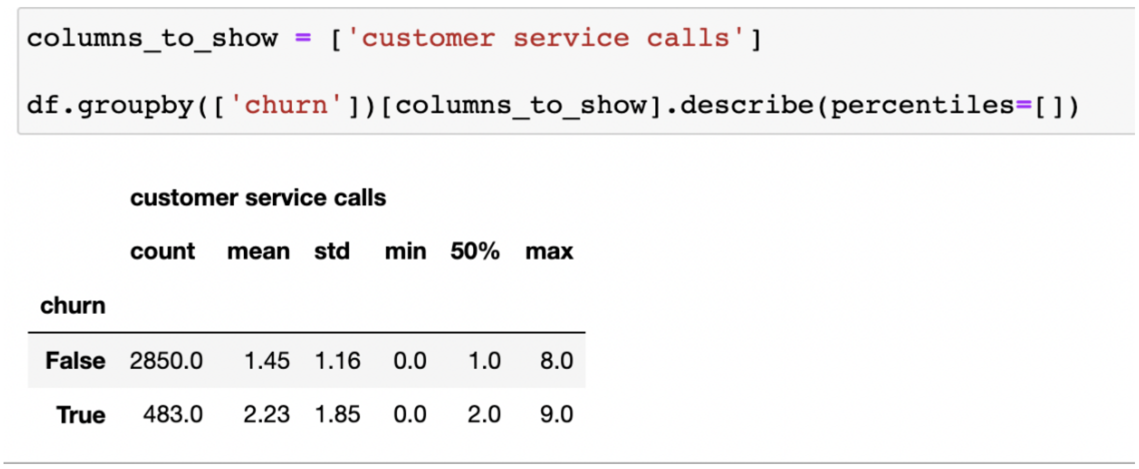


```
df.sort_values(by='total day charge', ascending=True).head(10)
```

rnational plan	voice mail plan	number vmail messages	total day minutes	total day calls	total day charge	total eve minutes	total eve calls	total eve charge	total night minutes	total night calls	total night charge	total intl minutes	total intl calls	total intl charge	customer service calls	churn	many_service_calls
no	no	0	0.0	0	0.00	159.6	130	13.57	167.1	88	7.52	6.8	1	1.84	4	True	1
no	no	0	0.0	0	0.00	192.1	119	16.33	168.8	95	7.60	7.2	4	1.94	1	False	0
no	yes	27	2.6	113	0.44	254.0	102	21.59	242.7	156	10.92	9.2	5	2.48	3	False	0
no	no	0	7.8	86	1.33	171.4	100	14.57	186.5	80	8.39	12.9	2	3.48	2	False	0
no	no	0	7.9	100	1.34	136.4	83	11.59	156.6	89	7.05	12.1	1	3.27	0	False	0
yes	no	0	12.5	67	2.13	256.6	90	21.81	169.4	88	7.62	7.7	9	2.08	1	False	0
no	no	0	17.6	121	2.99	161.7	125	13.74	203.1	82	9.14	10.6	6	2.86	1	False	0
no	no	0	18.9	92	3.21	258.4	81	21.96	109.6	74	4.93	14.8	4	4.00	1	False	0
no	yes	21	19.5	149	3.32	140.9	109	11.98	179.7	111	8.09	7.9	1	2.13	0	False	0
no	yes	27	25.9	119	4.40	206.5	96	17.55	228.1	64	10.26	6.5	7	1.76	1	False	0

While sorting in the ascending order by "total day charge", we can see that the churn is false is most of the cases.

10.



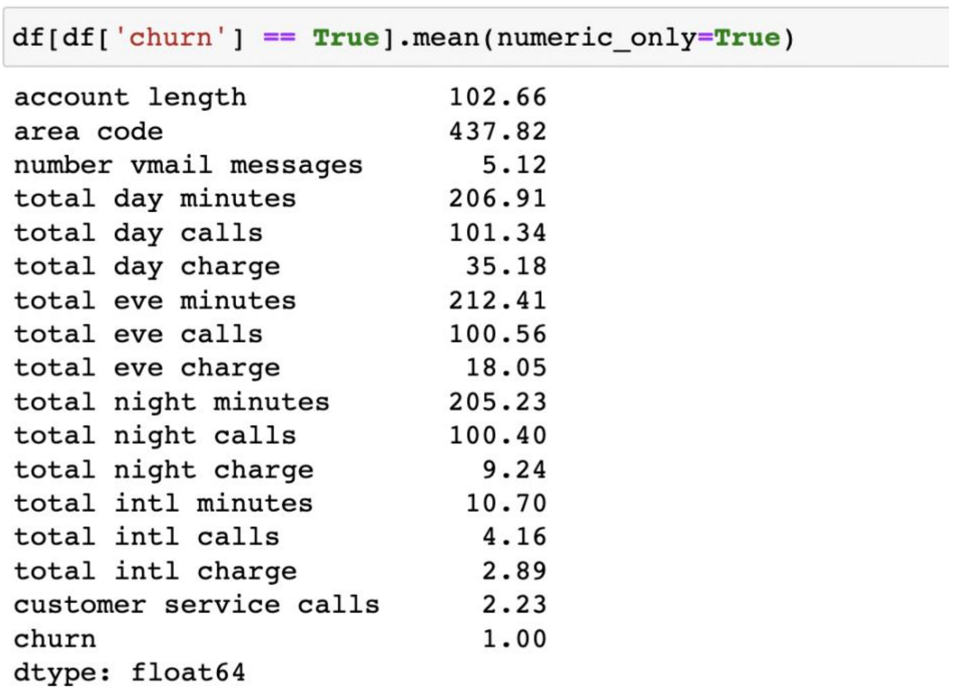
```
columns_to_show = ['customer service calls']

df.groupby(['churn'])[columns_to_show].describe(percentiles=[])
```

	customer service calls					
	count	mean	std	min	50%	max
churn						
False	2850.0	1.45	1.16	0.0	1.0	8.0
True	483.0	2.23	1.85	0.0	2.0	9.0

The average number of customer care calls made by customers who have left the company is 2.23, whereas the average number made by those who have s tayed is 1.45. This demonstrates that users who have left frequently make mor e calls to customer support while users who have stayed behind make less cal ls.

11.

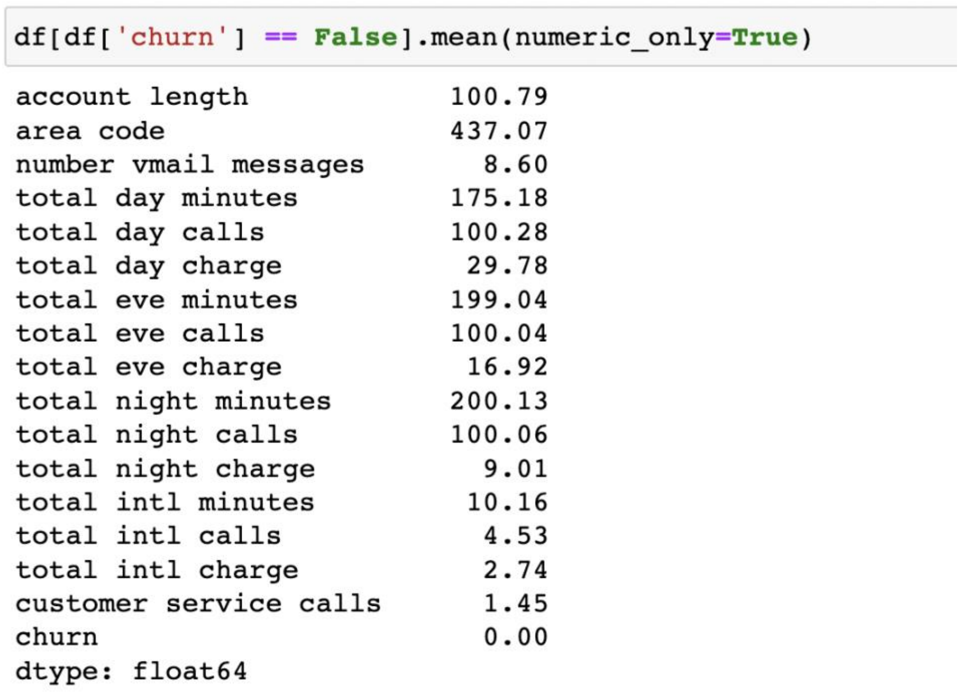
When churn is True,



```
df[df['churn'] == True].mean(numeric_only=True)
```

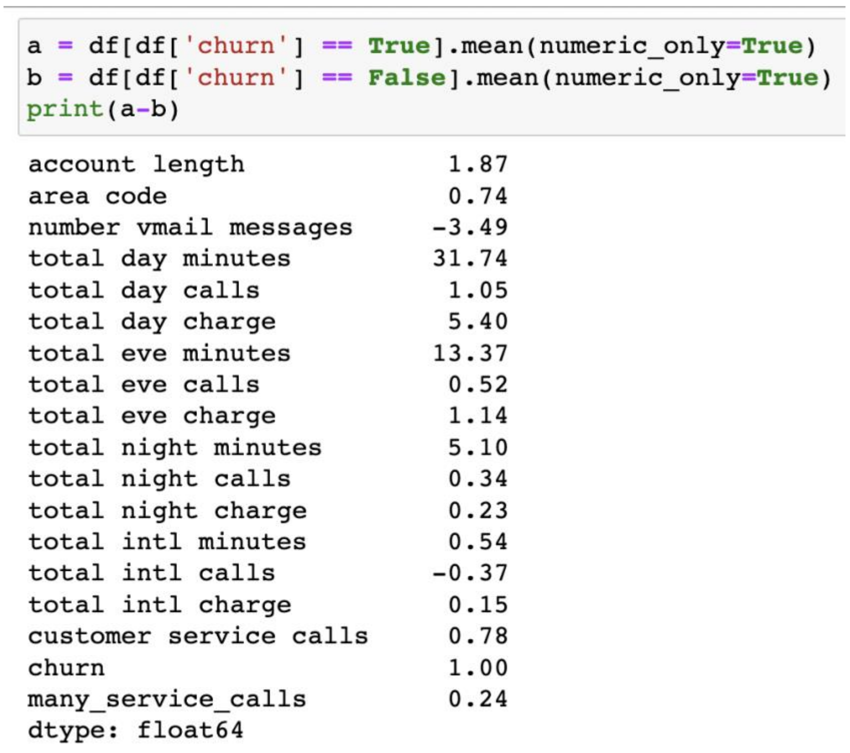
```
account length            102.66
area code                 437.82
number vmail messages       5.12
total day minutes         206.91
total day calls           101.34
total day charge           35.18
total eve minutes         212.41
total eve calls           100.56
total eve charge           18.05
total night minutes       205.23
total night calls         100.40
total night charge          9.24
total intl minutes         10.70
total intl calls            4.16
total intl charge           2.89
customer service calls      2.23
churn                       1.00
dtype: float64
```

When churn is False,



```
df[df['churn'] == False].mean(numeric_only=True)
```

```
account length            100.79
area code                 437.07
number vmail messages       8.60
total day minutes         175.18
total day calls           100.28
total day charge           29.78
total eve minutes         199.04
total eve calls           100.04
total eve charge           16.92
total night minutes       200.13
total night calls         100.06
total night charge          9.01
total intl minutes         10.16
total intl calls            4.53
total intl charge           2.74
customer service calls      1.45
churn                       0.00
dtype: float64
```

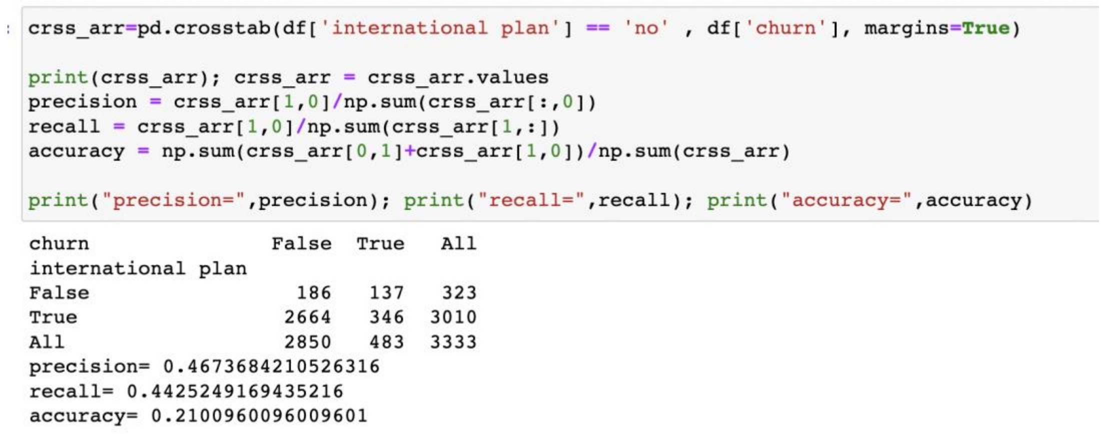


```
a = df[df['churn'] == True].mean(numeric_only=True)
b = df[df['churn'] == False].mean(numeric_only=True)
print(a-b)
```

```
account length              1.87
area code                   0.74
number vmail messages      -3.49
total day minutes          31.74
total day calls             1.05
total day charge            5.40
total eve minutes          13.37
total eve calls             0.52
total eve charge            1.14
total night minutes         5.10
total night calls           0.34
total night charge          0.23
total intl minutes          0.54
total intl calls           -0.37
total intl charge           0.15
customer service calls      0.78
churn                       1.00
many_service_calls          0.24
dtype: float64
```

Comparing both, we can observe that customers who left the company made roughly 1.5 times as many calls to customer service as those who stayed. This indicates that there may be a problem with customer care, such as delays in resolution, not being able to understand the customer's complaints, not prioritizing it etc. Also, those who have been "churned out" have fewer voice mail messages and fewer international calls than users who have stayed. Also, the total day minutes for the ones who have churned out is more indicating they need more talk time for the price than what the company is offering. So, in order to keep the people that are leaving, the company should develop a subscription that focuses on local calls, has more talk time, and is reasonably priced than what they are currently offering.

12.



```
crss_arr=pd.crosstab(df['international plan'] == 'no' , df['churn'], margins=True)

print(crss_arr); crss_arr = crss_arr.values
precision = crss_arr[1,0]/np.sum(crss_arr[:,0])
recall = crss_arr[1,0]/np.sum(crss_arr[1,:])
accuracy = np.sum(crss_arr[0,1]+crss_arr[1,0])/np.sum(crss_arr)

print("precision=",precision); print("recall=",recall); print("accuracy=",accuracy)
```

```
churn                False  True   All
international plan
False                  186   137   323
True                  2664   346  3010
All                   2850   483  3333
precision= 0.4673684210526316
recall= 0.4425249169435216
accuracy= 0.2100960096009601
```

When international call = no and churn is false, the performance metrics are as above.

13.

P(churn=True) = $\frac{483}{3333}$ = 0.145

P(churn=False) = $\frac{2850}{3333}$ = 0.855

P(International plan = yes) = $\frac{323}{3333}$ = 0.097

P(International plan = no) = $\frac{3010}{3333}$ = 0.903

P(churn = True| International plan = no) = $\frac{346}{3333}$ = 0.104

P(churn = False| International plan = no) = $\frac{2664}{3333}$ = 0.799

P(churn = True| International plan = yes) = $\frac{137}{3333}$ = 0.041

P(churn = False| International plan = yes) = $\frac{186}{3333}$ = 0.056

P(International plan = yes| churn = True) =

$$\frac{\mathrm{P}(\text{churn} = \text{True} \mid \text{International plan} = \text{yes}) * \mathrm{P}(\text{International plan=yes})}{\mathrm{P}(\text{churn=True})} = \frac{(0.041).(0.097)}{0.145} = 0.027$$

P(international plan = no| churn = True) =

$$\frac{\mathrm{P}(\text{churn=True|international plan=no}).\mathrm{P}(\text{international plan=no})}{\mathrm{P}(\text{churn=True})} = \frac{(0.104).(0.903)}{0.145} = 0.647$$

P(international plan = yes| churn = False) =

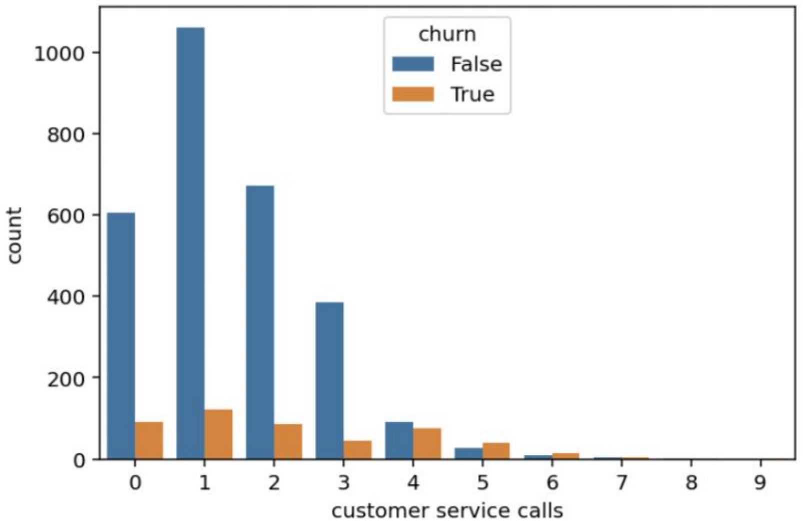
$$\frac{\mathrm{P}(\text{churn=False|international plan=yes}).\mathrm{P}(\text{international plan=yes})}{\mathrm{P}(\text{churn=False})} = \frac{(0.056).(0.097)}{0.855} = 0.006$$

P(international plan = no| churn = no) =

$$\frac{\mathrm{P}(\text{churn=False|international plan=no}).\mathrm{P}(\text{international plan=no})}{\mathrm{P}(\text{churn=False})} = \frac{(0.799).(0.903)}{0.855} = 0.844$$

14.

```
sns.countplot(x='customer service calls', hue='churn', data=df);
```



```
crss_arr=pd.crosstab(df['customer service calls'] == 0 , df['churn'], margins=True)

print(crss_arr); crss_arr = crss_arr.values
precision = crss_arr[1,1]/np.sum(crss_arr[:,1])
recall = crss_arr[1,1]/np.sum(crss_arr[1,:])
accuracy = np.sum(crss_arr[0,0]+crss_arr[1,1])/np.sum(crss_arr)

print("precision=",precision); print("recall=",recall); print("accuracy=",accuracy)
```

```
churn                   False  True   All
customer service calls
False                    2245   391  2636
True                      605    92   697
All                      2850   483  3333
precision= 0.09523809523809523
recall= 0.06599713055954089
accuracy= 0.1752925292529253
```

P (churn = True | customer service calls = 0) =

$$\frac{P(cust\ ser\ call=0\mid churn=True)*P(churn=True)}{P(cust\ serv\ calls.=0)} = \frac{\frac{92}{3333}*\frac{483}{3333}}{\frac{697}{3333}} = (92 * 483) / (697* 3333) =$$

0.02

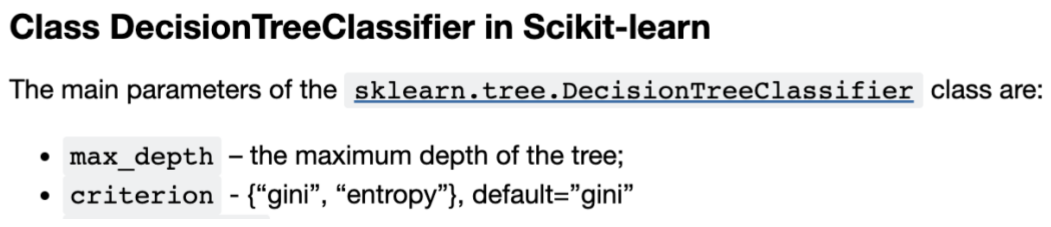
15.

```
crss_arr=pd.crosstab(df['many_service_calls'] & (df['international plan'] == 'yes') , df['churn'], margins=True)

print(crss_arr); crss_arr = crss_arr.values
precision = crss_arr[1,1]/np.sum(crss_arr[:,1])
recall = crss_arr[1,1]/np.sum(crss_arr[1,:])
accuracy = np.sum(crss_arr[0,0]+crss_arr[1,1])/np.sum(crss_arr)

print("precision=",precision); print("recall=",recall); print("accuracy=",accuracy)
```

```
churn  False  True   All
row_0
False   2841   464  3305
True       9    19    28
All     2850   483  3333
precision= 0.019668737060041408
recall= 0.3392857142857143
accuracy= 0.2145214521452145
```

Experiment 2:

1.



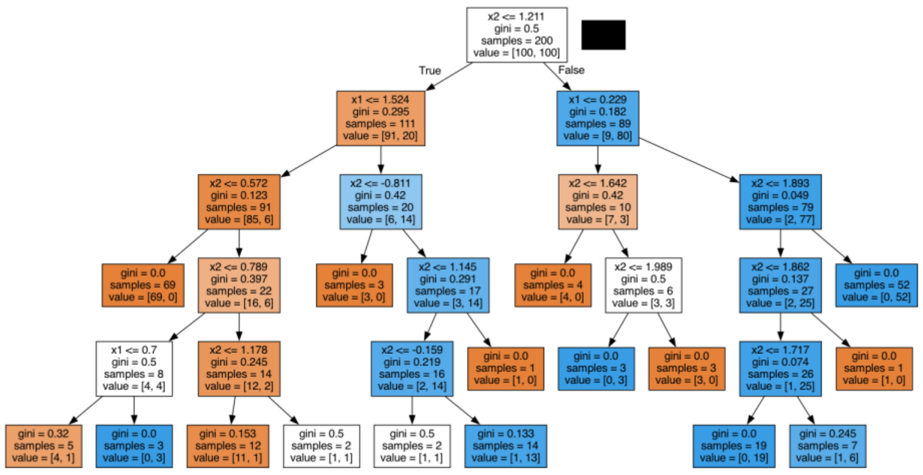
Class DecisionTreeClassifier in Scikit-learn

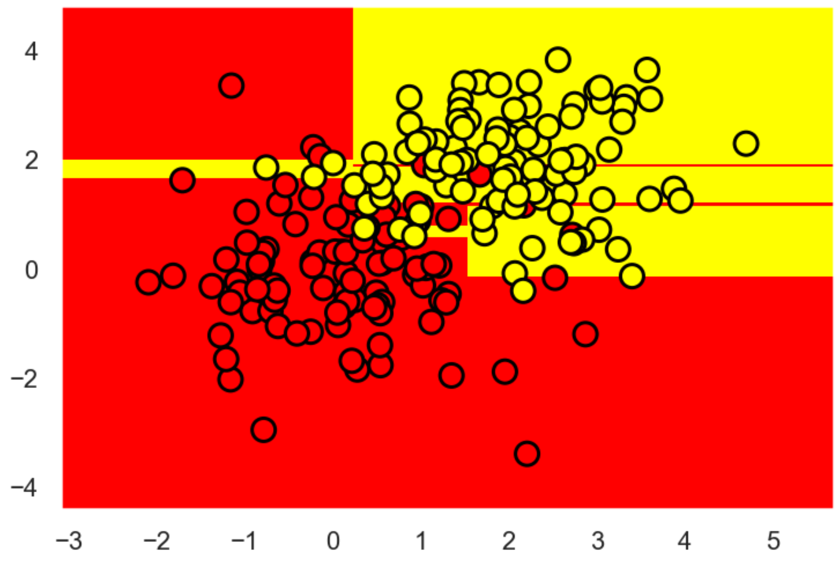
The main parameters of the `sklearn.tree.DecisionTreeClassifier` class are:

- `max_depth` – the maximum depth of the tree;
- `criterion` - {“gini”, “entropy”}, default=”gini”

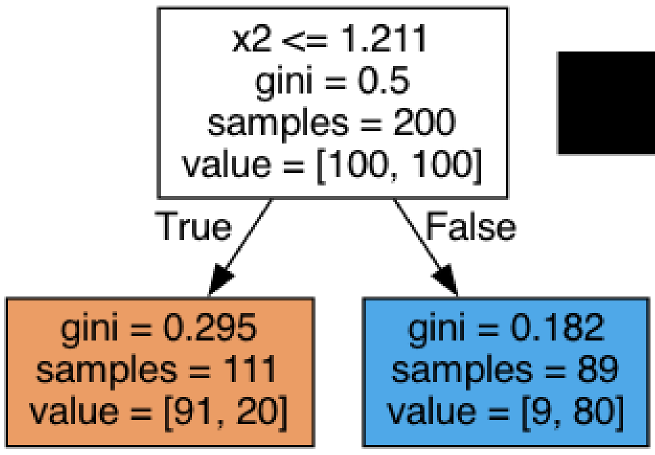
Nodes are expanded until all leaves are pure if max depth is None. If criterion is not defined, by default, gini in considered.

2. When overfitting i.e. max_depth = 5 (Considering 5 since >= 4)

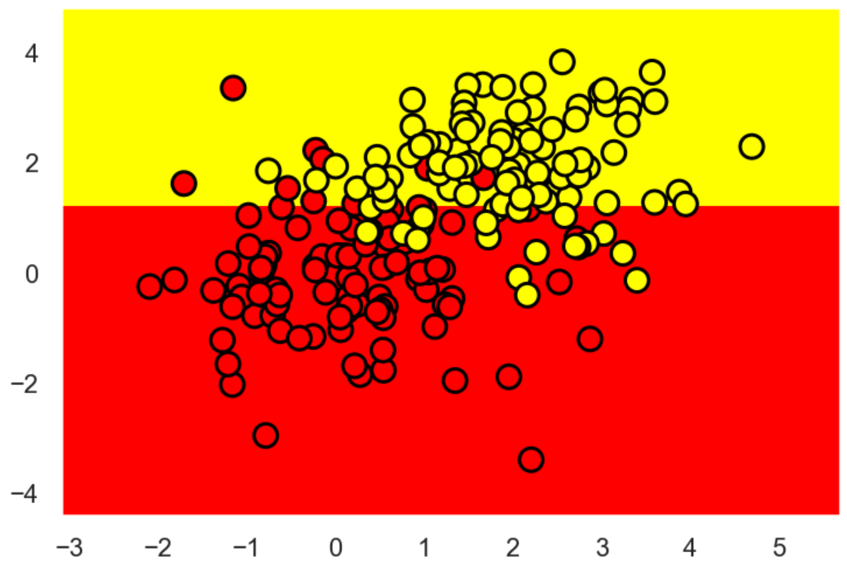

[redacted]
x2 <= 1.211 gini = 0.5 samples = 200 value = [100, 100]
True
False
x1 <= 1.524 gini = 0.295 samples = 111 value = [91, 20]
x1 <= 0.229 gini = 0.182 samples = 89 value = [9, 80]
x2 <= 0.572 gini = 0.123 samples = 91 value = [85, 6]
x2 <= -0.811 gini = 0.42 samples = 20 value = [6, 14]
x2 <= 1.642 gini = 0.42 samples = 10 value = [7, 3]
x2 <= 1.893 gini = 0.049 samples = 79 value = [2, 77]
gini = 0.0 samples = 69 value = [69, 0]
x2 <= 0.789 gini = 0.397 samples = 22 value = [16, 6]
gini = 0.0 samples = 3 value = [3, 0]
x2 <= 1.145 gini = 0.291 samples = 17 value = [3, 14]
gini = 0.0 samples = 4 value = [4, 0]
x2 <= 1.989 gini = 0.5 samples = 6 value = [3, 3]
x2 <= 1.862 gini = 0.137 samples = 27 value = [2, 25]
gini = 0.0 samples = 52 value = [0, 52]
x1 <= 0.7 gini = 0.5 samples = 8 value = [4, 4]
x2 <= 1.178 gini = 0.245 samples = 14 value = [12, 2]
x2 <= -0.159 gini = 0.219 samples = 16 value = [2, 14]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.0 samples = 3 value = [0, 3]
gini = 0.0 samples = 3 value = [3, 0]
x2 <= 1.717 gini = 0.074 samples = 26 value = [1, 25]
gini = 0.0 samples = 1 value = [1, 0]
gini = 0.32 samples = 5 value = [4, 1]
gini = 0.0 samples = 3 value = [0, 3]
gini = 0.153 samples = 12 value = [11, 1]
gini = 0.5 samples = 2 value = [1, 1]
gini = 0.5 samples = 2 value = [1, 1]
gini = 0.133 samples = 14 value = [1, 13]
gini = 0.0 samples = 19 value = [0, 19]
gini = 0.245 samples = 7 value = [1, 6]



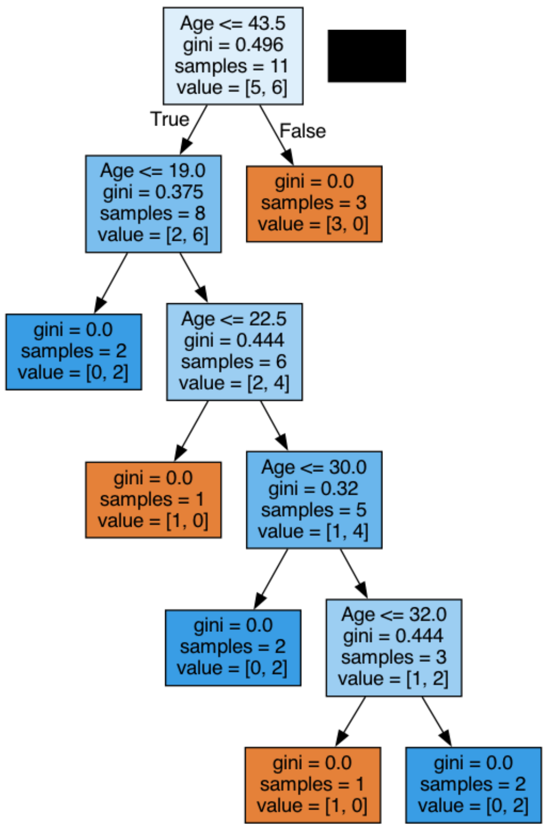
Similarly, when considering underfitting i.e. max_depth = 1



[redacted]



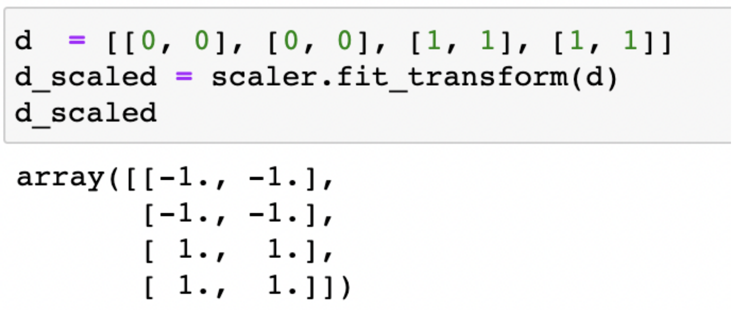
3. For the bank dataset, the tree is as below



[redacted]

The tree used the following 5 values to evaluate by age: 43.5, 19, 22.5, 30, and 32 years. These are exactly the mean values between the ages at which the target class "switches" from 1 to 0 or 0 to 1. For example, 43.5 is the average of 38 and 49 years; a 38-year-old customer failed to return the loan whereas the 49-year-old did. The tree looks for the values at which the target class switches its value as a threshold for "cutting" a quantitative variable.

4.
The standard scalar standardizes the features by eliminating the mean and sc aling to unit variance.



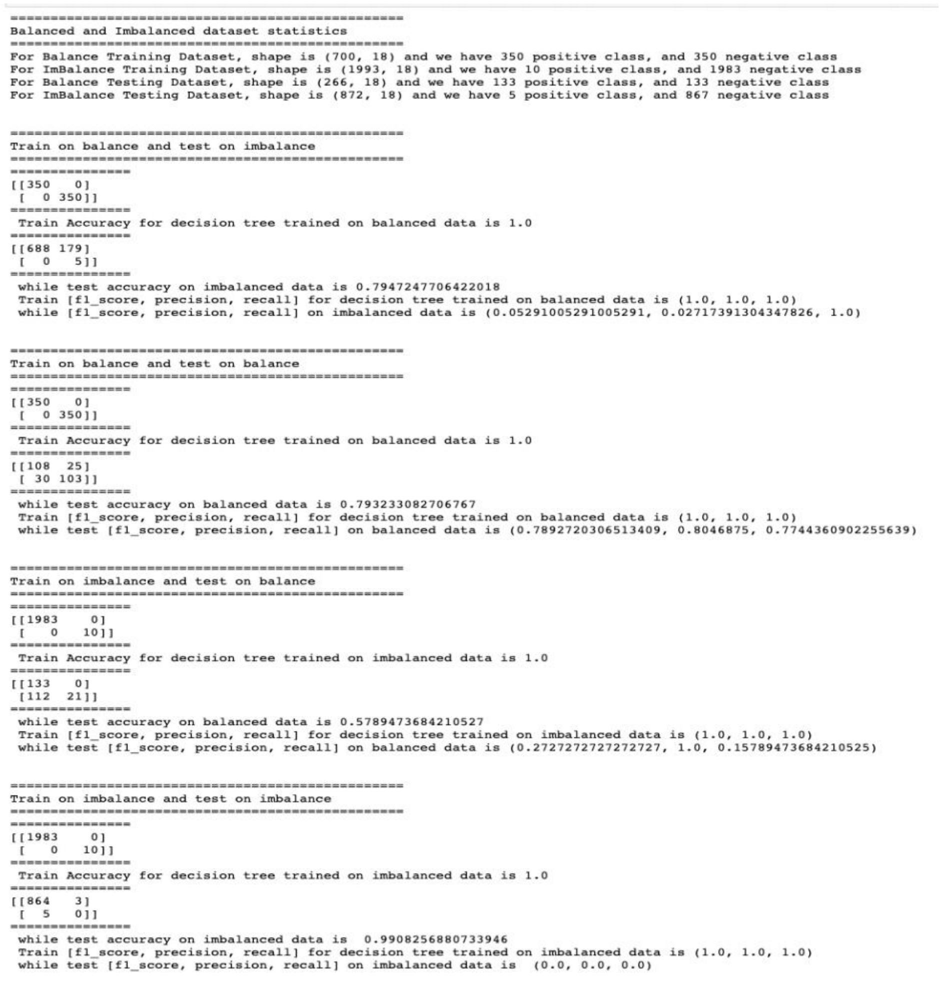
```
d  = [[0, 0], [0, 0], [1, 1], [1, 1]]
d_scaled = scaler.fit_transform(d)
d_scaled
```

```
array([[-1., -1.],
       [-1., -1.],
       [ 1.,  1.],
       [ 1.,  1.]])
```

The scalar transforms are ([[-1., -1.], [-1., -1.], [1., 1.], [1., 1.]]).

5. Under-fitting typically results from small decision trees, while over-fitting typically results from large decision trees. Using both little and large amounts of data, the training and testing accuracy for overfitting and underfitting decision tree models is done. Overfit models have high accuracy for training data but low accuracy for test data. Since the big data has a higher variance and lower bias than small data, test data accuracy in overfit models is a little bit higher in big data than in small data. This is what is observer in the experiment. Similarly, due to its large bias and low variance, the underfit model has poor accuracy for both training and test data, regardless of the size of the data set.

6. The performance metrics of the balanced and unbalanced data is displayed in the experiment. The model is first trained on a balanced dataset and tested on an unbalanced dataset. The same procedure is carried out for other combinations as well. This includes training and testing on a balanced dataset, training on an unbalanced dataset and testing on a balanced dataset, and training and testing on an unbalanced dataset. The set is divided into 30% for validation and 70% for training. or each of the above-mentioned case, a confusion matrix is constructed. Then, using the built-in Sklearn functions, the appropriate F1 score, precision, and recall are observed as seen below.



```
==================================================
Balanced and Imbalanced dataset statistics
==================================================
For Balance Training Dataset, shape is (700, 18) and we have 350 positive class, and 350 negative class
For ImBalance Training Dataset, shape is (1993, 18) and we have 10 positive class, and 1983 negative class
For Balance Testing Dataset, shape is (266, 18) and we have 133 positive class, and 133 negative class
For ImBalance Testing Dataset, shape is (872, 18) and we have 5 positive class, and 867 negative class


==================================================
Train on balance and test on imbalance
==================================================
===============
[[350   0]
 [  0 350]]
===============
 Train Accuracy for decision tree trained on balanced data is 1.0
===============
[[688 179]
 [  0   5]]
===============
 while test accuracy on imbalanced data is 0.7947247706422018
 Train [f1_score, precision, recall] for decision tree trained on balanced data is (1.0, 1.0, 1.0)
 while [f1_score, precision, recall] on imbalanced data is (0.05291005291005291, 0.02717391304347826, 1.0)


==================================================
Train on balance and test on balance
==================================================
===============
[[350   0]
 [  0 350]]
===============
 Train Accuracy for decision tree trained on balanced data is 1.0
===============
[[108  25]
 [ 30 103]]
===============
 while test accuracy on balanced data is 0.793233082706767
 Train [f1_score, precision, recall] for decision tree trained on balanced data is (1.0, 1.0, 1.0)
 while test [f1_score, precision, recall] on balanced data is (0.7892720306513409, 0.8046875, 0.7744360902255639)


==================================================
Train on imbalance and test on balance
==================================================
===============
[[1983    0]
 [   0   10]]
===============
 Train Accuracy for decision tree trained on imbalanced data is 1.0
===============
[[133   0]
 [112  21]]
===============
 while test accuracy on balanced data is 0.5789473684210527
 Train [f1_score, precision, recall] for decision tree trained on imbalanced data is (1.0, 1.0, 1.0)
 while test [f1_score, precision, recall] on balanced data is (0.2727272727272727, 1.0, 0.15789473684210525)


==================================================
Train on imbalance and test on imbalance
==================================================
===============
[[1983    0]
 [   0   10]]
===============
 Train Accuracy for decision tree trained on imbalanced data is 1.0
===============
[[864   3]
 [  5   0]]
===============
 while test accuracy on imbalanced data is  0.9908256880733946
 Train [f1_score, precision, recall] for decision tree trained on imbalanced data is (1.0, 1.0, 1.0)
 while test [f1_score, precision, recall] on imbalanced data is  (0.0, 0.0, 0.0)
```

	F1 score	Precision	Recall
Training balanced	1.0	1.0	1.0
Testing imbalanced	0.05291005291005291	0.02717391304347826	1.0
Training balanced	1.0	1.0	1.0
Testing balanced	0.7892720306513409	0.8046875	0.7744360902255639
Training imbalanced	1.0	1.0	1.0
Testing balanced	0.2727272727272727	1.0	0.15789473684210525
Training imbalanced	1.0	1.0	1.0
Testing imbalanced	0.0	0.0	0.0

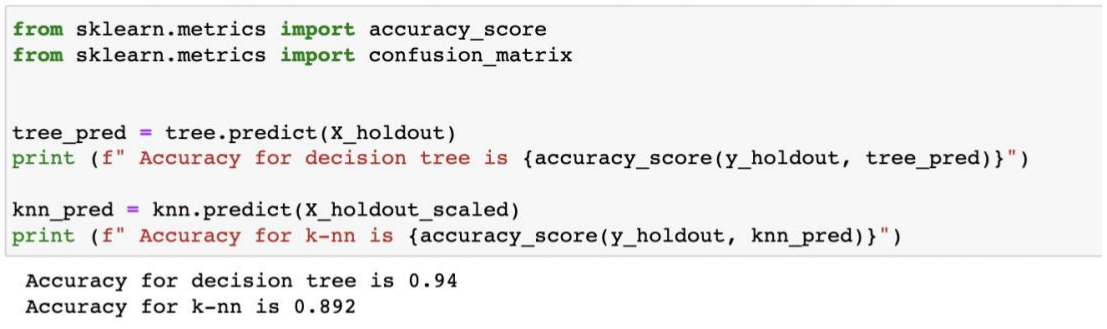
Experiments' precision, recall, and f1 score

7.

The decision tree model was applied to the dataset in the experiment, and the results are verified after some noise i.e. irrelevant attributes are added to both the test and training sets. Due to the model being overfit, adding irrelevant features decreases the accuracy of the model over the test dataset.

8.

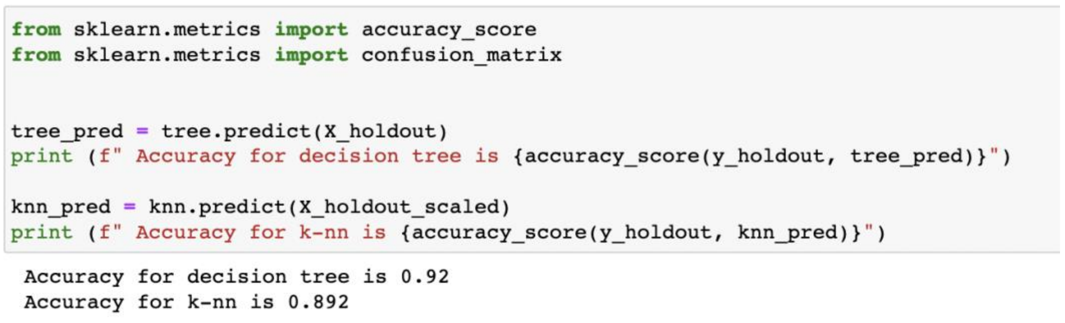
When max_depth = 6, accuracy is 94% as below



```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


tree_pred = tree.predict(X_holdout)
print (f" Accuracy for decision tree is {accuracy_score(y_holdout, tree_pred)}")

knn_pred = knn.predict(X_holdout_scaled)
print (f" Accuracy for k-nn is {accuracy_score(y_holdout, knn_pred)}")
```

```
 Accuracy for decision tree is 0.94
 Accuracy for k-nn is 0.892
```

When max_depth = None, accuracy = 92%

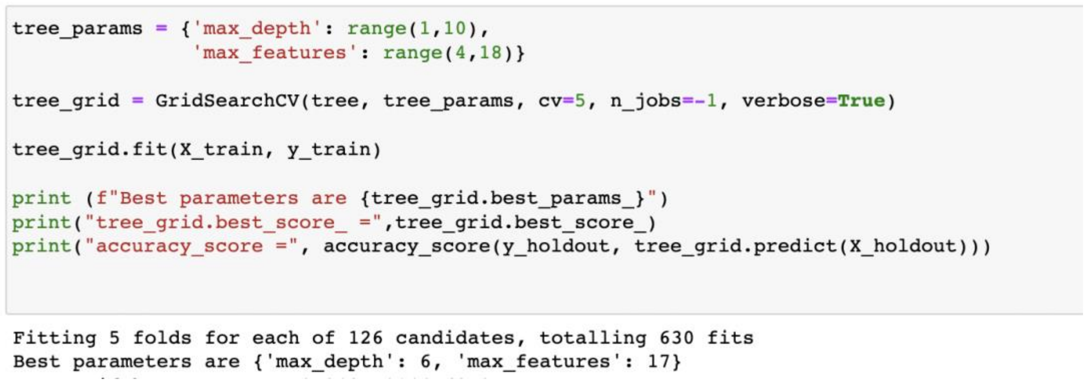


```
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


tree_pred = tree.predict(X_holdout)
print (f" Accuracy for decision tree is {accuracy_score(y_holdout, tree_pred)}")

knn_pred = knn.predict(X_holdout_scaled)
print (f" Accuracy for k-nn is {accuracy_score(y_holdout, knn_pred)}")
```

```
 Accuracy for decision tree is 0.92
 Accuracy for k-nn is 0.892
```

When max depth is set to the default value, the model overfits. This results in a loss in accuracy. As previously mentioned, the tree grows until the leaf nodes achieve their pure state when max depth is set to default.

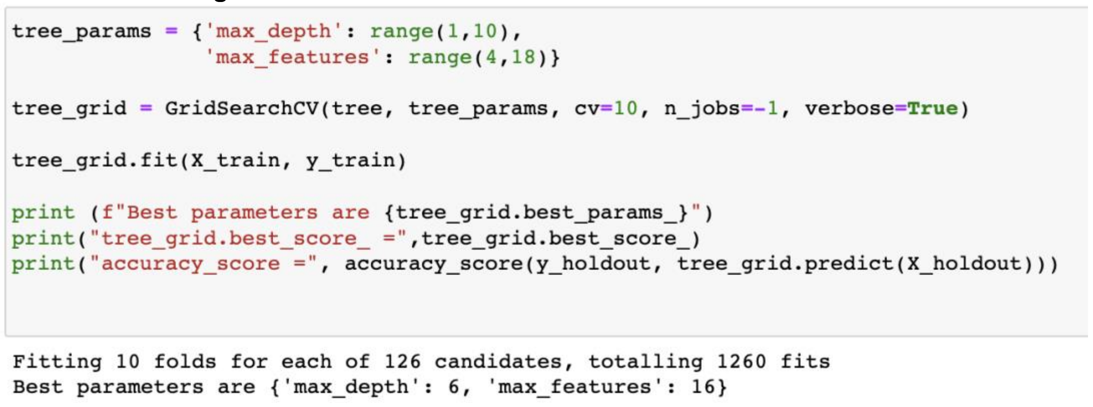
9. By default, we can fit 630 decision trees.



```
tree_params = {'max_depth': range(1,10),
               'max_features': range(4,18)}

tree_grid = GridSearchCV(tree, tree_params, cv=5, n_jobs=-1, verbose=True)

tree_grid.fit(X_train, y_train)

print (f"Best parameters are {tree_grid.best_params_}")
print("tree_grid.best_score_ =",tree_grid.best_score_)
print("accuracy_score =", accuracy_score(y_holdout, tree_grid.predict(X_holdout)))
```

```
Fitting 5 folds for each of 126 candidates, totalling 630 fits
Best parameters are {'max_depth': 6, 'max_features': 17}
```

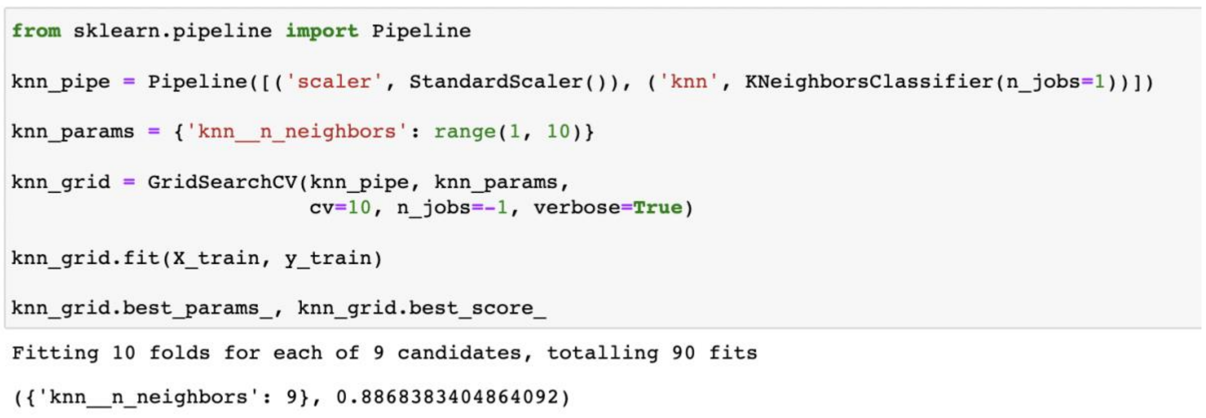
When we change cross validation value to 10, we can fit 1260 decision trees.



```
tree_params = {'max_depth': range(1,10),
               'max_features': range(4,18)}

tree_grid = GridSearchCV(tree, tree_params, cv=10, n_jobs=-1, verbose=True)

tree_grid.fit(X_train, y_train)

print (f"Best parameters are {tree_grid.best_params_}")
print("tree_grid.best_score_ =",tree_grid.best_score_)
print("accuracy_score =", accuracy_score(y_holdout, tree_grid.predict(X_holdout)))
```

```
Fitting 10 folds for each of 126 candidates, totalling 1260 fits
Best parameters are {'max_depth': 6, 'max_features': 16}
```

10.
K = 9 is the best choice.

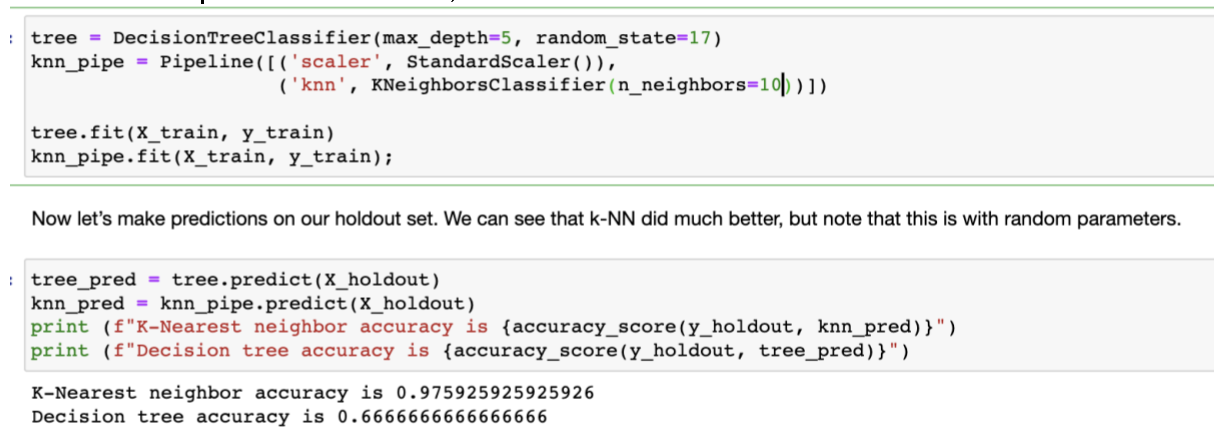


```
from sklearn.pipeline import Pipeline

knn_pipe = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifier(n_jobs=1))])

knn_params = {'knn__n_neighbors': range(1, 10)}

knn_grid = GridSearchCV(knn_pipe, knn_params,
                        cv=10, n_jobs=-1, verbose=True)

knn_grid.fit(X_train, y_train)

knn_grid.best_params_, knn_grid.best_score_
```

```
Fitting 10 folds for each of 9 candidates, totalling 90 fits

({'knn__n_neighbors': 9}, 0.8868383404864092)
```

11.

When maxdepth = 5 and k =10,

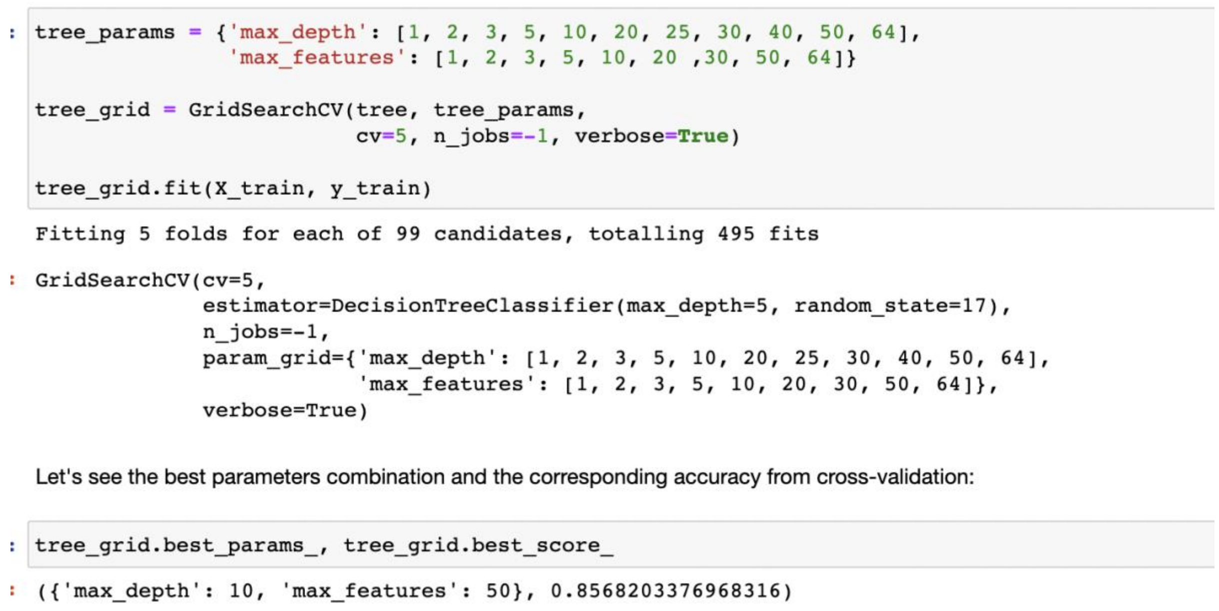


```
tree = DecisionTreeClassifier(max_depth=5, random_state=17)
knn_pipe = Pipeline([('scaler', StandardScaler()),
                     ('knn', KNeighborsClassifier(n_neighbors=10))])

tree.fit(X_train, y_train)
knn_pipe.fit(X_train, y_train);
```

Now let's make predictions on our holdout set. We can see that k-NN did much better, but note that this is with random parameters.

```
tree_pred = tree.predict(X_holdout)
knn_pred = knn_pipe.predict(X_holdout)
print (f"K-Nearest neighbor accuracy is {accuracy_score(y_holdout, knn_pred)}")
print (f"Decision tree accuracy is {accuracy_score(y_holdout, tree_pred)}")
```

```
K-Nearest neighbor accuracy is 0.975925925925926
Decision tree accuracy is 0.6666666666666666
```



```
tree_params = {'max_depth': [1, 2, 3, 5, 10, 20, 25, 30, 40, 50, 64],
               'max_features': [1, 2, 3, 5, 10, 20 ,30, 50, 64]}

tree_grid = GridSearchCV(tree, tree_params,
                         cv=5, n_jobs=-1, verbose=True)

tree_grid.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 99 candidates, totalling 495 fits
```

```
GridSearchCV(cv=5,
             estimator=DecisionTreeClassifier(max_depth=5, random_state=17),
             n_jobs=-1,
             param_grid={'max_depth': [1, 2, 3, 5, 10, 20, 25, 30, 40, 50, 64],
                         'max_features': [1, 2, 3, 5, 10, 20, 30, 50, 64]},
             verbose=True)
```

Let's see the best parameters combination and the corresponding accuracy from cross-validation:

```
tree_grid.best_params_, tree_grid.best_score_
```

```
({'max_depth': 10, 'max_features': 50}, 0.8568203376968316)
```

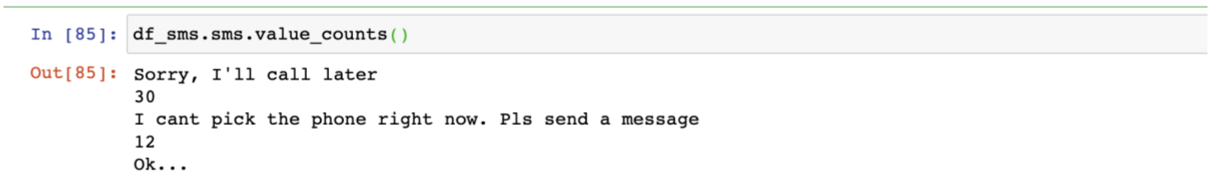
When 5-fold cross validation, best params are max depth 10 and max features 50. The accuracy is 85%.

Experiment 3:

1. There are 5572 records in the data set. The distribution of the labels is
   ham 4825
   spam 747

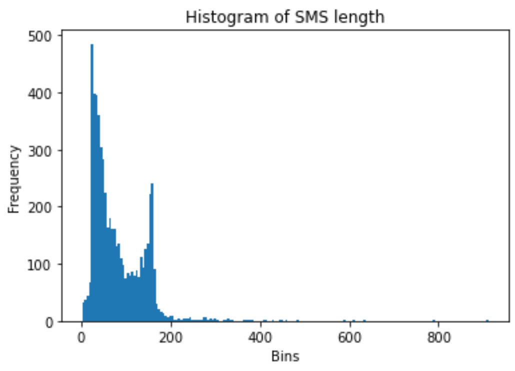
   ham is right skewed. Spam is left skewed. The histogram of sms length is also right skewed.

2.
   5169 unique sms message. Most frequent sms message is “Sorry, I'll call later”. Its frequency is 30.

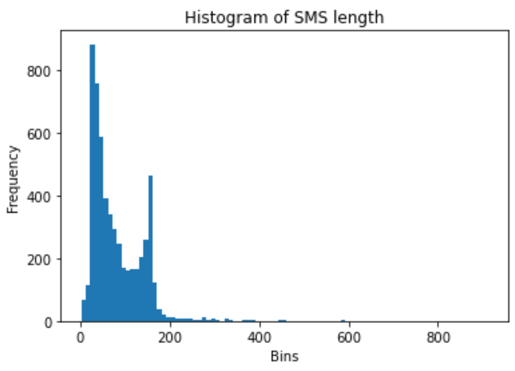


```
In [85]: df_sms.sms.value_counts()

Out[85]: Sorry, I'll call later
         30
         I cant pick the phone right now. Pls send a message
         12
         Ok...
```

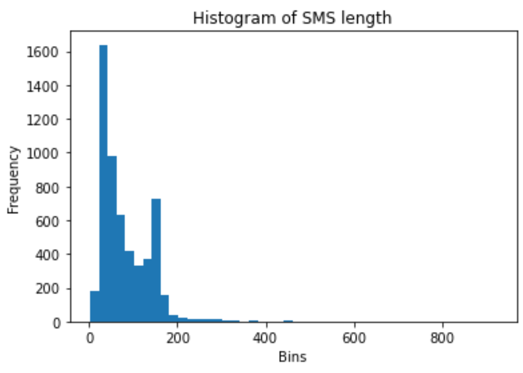
3. Maximum length of SMS is 910. Minimum length of SMS is 2.



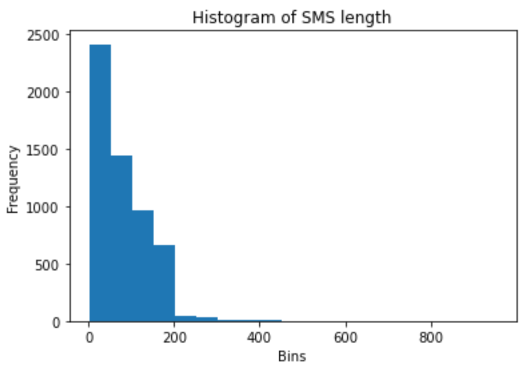
Bin size - 5



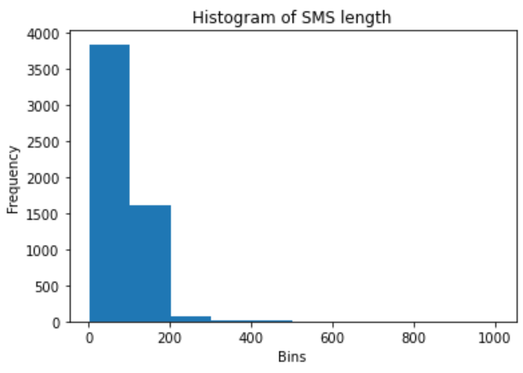
Bin size - 10



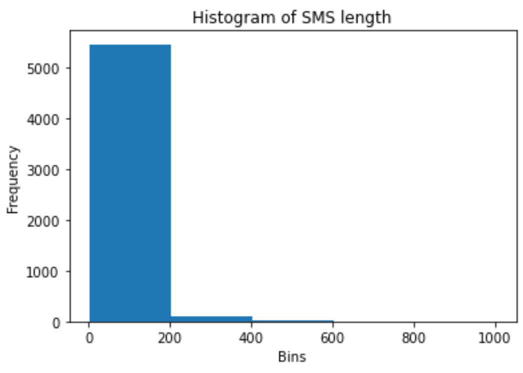
Bin size – 20



Bin size -50



Bin size – 100

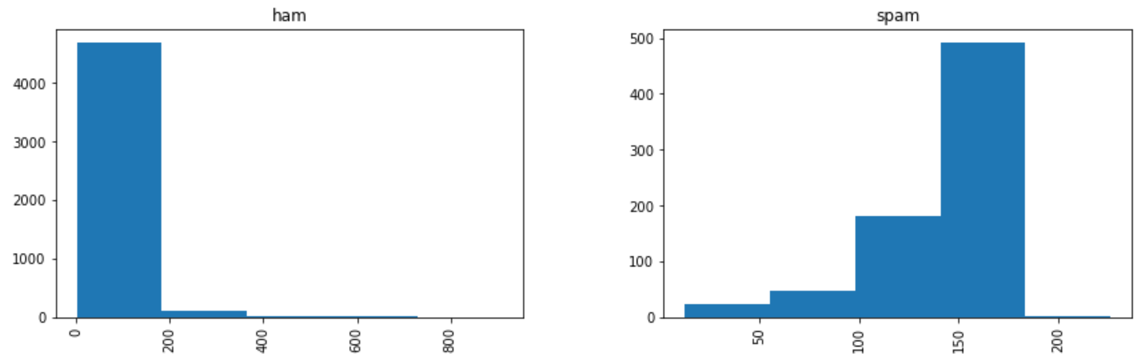


Bin size – 200

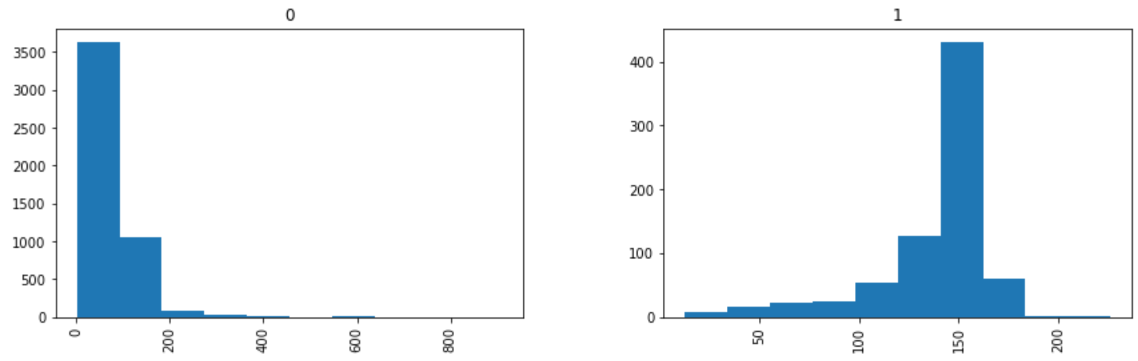
The bin width affects the ability of a histogram to identify local regions of higher incidence. So as it increases, the histogram will lack the details needed to discern any useful pattern from the data. In particular here, we can observe that as the size increases, we are losing any indication of a second peak.

4.

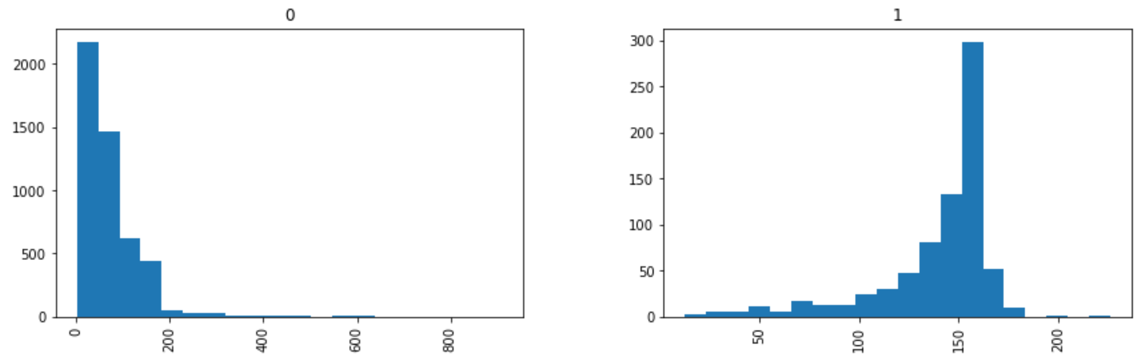
5 Bins



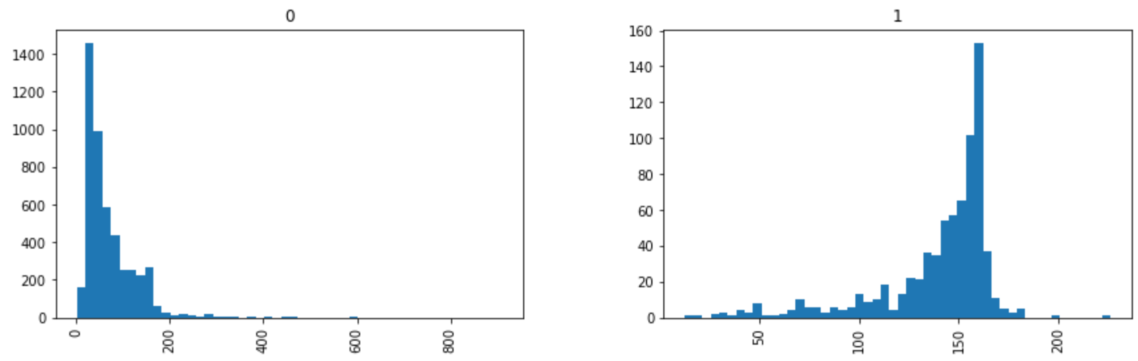
10 Bin



20 Bins



50 Bins



With a decrease in bin width, there will be more bins. So, the quality of the data increases with the number of bins.

5. We convert all of them into lower case to normalize all the words so that it will be easier to get the frequency. For example, “who” and “Who” are the same words, if we don’t normalize it, it will consider them as 2 different words. So we convert all the words in the documents to their lower cases. Yes, we can convert all the words into upper case also. As long as it is normalized in specific order, it is fine. But a combination is not allowed.

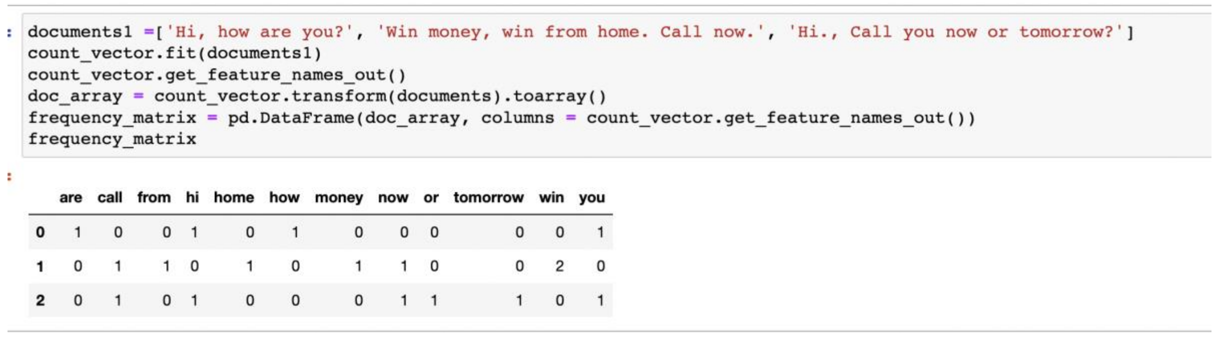
6. Count vectorization does the following:
    a) It tokenizes the string(separates the string into individual words) and gives an integer ID to each token.
    b) It then counts the occurrence of each of those tokens.

    If set to English will remove all words from our document set that match a list of English stop words which is defined in scikit-learn. a, an, the, is, has are 5 stop words in English defined in scikit-learn.

7. We first initialize the Count Vectorizer method and then fit the training data using “fit_transform” to get the document matrix.

    We first separate the data into train/test and then generate a document-term matrix based on the training dataset and afterward generate a matrix for the test set.

8.



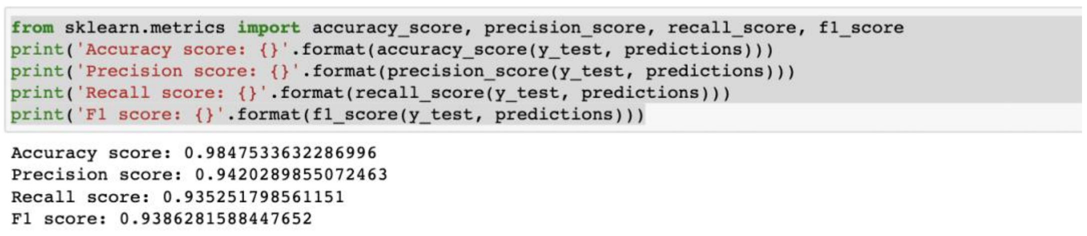
```
: documents1 =['Hi, how are you?', 'Win money, win from home. Call now.', 'Hi., Call you now or tomorrow?']
  count_vector.fit(documents1)
  count_vector.get_feature_names_out()
  doc_array = count_vector.transform(documents).toarray()
  frequency_matrix = pd.DataFrame(doc_array, columns = count_vector.get_feature_names_out())
  frequency_matrix
:
```

	are	call	from	hi	home	how	money	now	or	tomorrow	win	you
0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	1	1	0	1	0	1	1	0	0	2	0
2	0	1	0	1	0	0	0	1	1	1	0	1

9. There are 7777 features created while making the document term matrix for the SMS dataset.

   To decrease the number of features, the features must be combined or removed for being irrelevant. The advantage is that the model's complexity reduces, while the drawback is that after training, certain features might still play a role in the model's settings.

10. Since our input data is discrete (word counts for text classification), we should use multinomial Naive Bayes implementation as it is suitable for classification with discrete features



```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
print('Accuracy score: {}'.format(accuracy_score(y_test, predictions)))
print('Precision score: {}'.format(precision_score(y_test, predictions)))
print('Recall score: {}'.format(recall_score(y_test, predictions)))
print('F1 score: {}'.format(f1_score(y_test, predictions)))
```

```
Accuracy score: 0.9847533632286996
Precision score: 0.9420289855072463
Recall score: 0.935251798561151
F1 score: 0.9386281588447652
```

Experiment- 4 : Diabetes data

Goal: The goal of the diabetes dataset is to determine whether a person has diabetes based on characteristics like age, pregnancy history, blood pressure, skin thickness, insulin levels, and diabetes degree of function.

Content: The input dataset has 9 columns and 768 records; the first 8 columns contain the dataset's features, while the final column has the label. The dataset contains 268 patients with diabetes and 500 patients without diabetes, and the output label is a binary datatype. Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, and Age are attributes that have discrete data types, whereas BMI and DiabetesPedigreeFunction have continuous data types.

Missing values: The given dataset does not contain any missing or null data.

Outliers in the dataset: The following attributes are among the few outliers in this dataset that have a value of 0. Blood pressure has 35 zeros(19 people do not have diabetes and 16 people have it). Similarly, glucose has 5 zeros(3 people do not have diabetes and 2 people do). For skin thickness, it has 227 zeros(139 people do not have diabetes and 88 people have it). SImilarly, the BMI has 11 zeros(9 people do not have diabetes and 2 people have diabetes) and insulin has 374 zeros(236 people do not have diabetes and 138 people have diabetes). Where BloodPressure, BMI, and Glucose have no zeros(724 records).

Feature: Features like glucose, skin thickness, age, diabetes pedigree function, and insulin have variances in their plots compared to other features in the histogram data grouped between individual feature and outcome. As a result, these features have a significant impact during training the model.

Testing Strategy: Testing is done using a stratified K Fold with 10 splits. The classification accuracy of algorithms is as below:

Using Train/ Test split:

Model	Accuracy
KNN	0.729282
DB	0.745856
GNB	0.734807
BNB	0.657459

Using K-Fold cross validation:

Model	Accuracy
KNN	0.714136
DB	0.696328
GNB	0.754205
BNB	0.656069

Although DB has the best accuracy in the train/test split, its accuracy in the K-Fold cross validation is lower because the model is overfitted as a result of the train/test split's lower variance than the K-Fold cross validation. Both the Train/Test Split and K-means cross validation demonstrate consistent accuracy for GNB. A better model for the diabetes data would therefore be GNB.

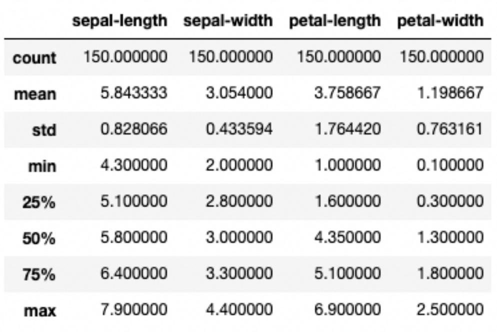
Experiment- 4 : Iris data

Goal: Using parameters like "sepal length, sepal breadth, petal length, and petal width," the iris dataset aims to identify diabetes in a person.

Content: The input dataset consists of 150 records in 5 columns, with the last column containing the label and the first 4 columns representing the dataset's features. The dataset includes outcomes for 50 Iris-setosa, 50 Iris-versicolor, and 50 Iris-virginica, and the output label is a categorical datatype. Every feature uses continuous data.

Missing values: The given dataset does not contain any missing or null data.

Feature: All features have variations in their plots compared to other features, which means that all of these characteristics have a significant impact on training the model as seen in the histogram data grouped between individual feature and outcome.



	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

Testing Strategy: Testing is done using a stratified K Fold with 10 splits. The classification accuracy of algorithms is as below:

Using Train/ Test split:

Model	Accuracy
KNN	1.000000
DB	0.973684
GNB	0.315789
BNB	0.973684

Using K-Fold cross validation:

Model	Accuracy
KNN	0.966667
DB	0.953333
GNB	0.333333
BNB	0.953333

In both the train/test split and K-Fold validation, KNN has the maximum accuracy. KNN would therefore be a superior model for Iris data.

Experiment- 4 : Thyroid data

Goal: The goal of the thyroid dataset is to determine whether a person has a thyroid gland based on attributes such T3 resin, Serum thyroxin, Serum triiodothyronine, Basal TSH, and Abs diff TSH.

Content: The input dataset consists of 215 entries in 6 columns, with the first 5 columns representing the dataset's features and the last column containing a label. The dataset contains 150 patients with thyroid type "1," 35 patients with thyroid type "2," and 30 patients with thyroid type "3," and the output label is a categorical datatype with three classes (1, 2, and 3).

Missing values: The given dataset does not contain any missing or null data.

Feature: All features have changes in their plots compared to other features, which means that all of these characteristics have a significant impact on training the model, according to the histogram data grouped between individual feature and outcome.

Testing Strategy: Testing is done using a stratified K Fold with 10 splits. The classification accuracy of algorithms is as below:

Using Train/Test split:

Model	Accuracy
KNN	0.944444
DB	0.888889
GNB	0.944444
BNB	0.796296

Using K-Fold cross validation:

Model	Accuracy
KNN	0.925758
DB	0.939394
GNB	0.967749
BNB	0.734848

Conclusion: Although GNB has greater accuracy in the K-Fold validation, KNN and GNB have the highest accuracy in the train/test split. GNB would therefore be a better model for thyroid data.